Merge Sort

Introduction

Merge sort works as two steps. First, divide the unsorted list into n sublists, each containing one element (a list of one element is considered sorted). Then, merge sublists to produce new sorted sublists repeatedly until there is only one sublist remaining. This will be the sorted list [4].

Performance

In general, the performance of the sorting algorithm is analyzed from two aspects, time complexity and space complexity. Time complexity represents the time taken to execute the algorithm, which is generally considered in three cases: best case, worst case, and average case [6]. Space complexity represents the amount of memory required to complete a program [5].

Use **array** as data structure and **n** denotes the input array size, merge sort performance is as follows [4]:

Worst-case time complexity	$\boldsymbol{O(n\ log\ n)}$
Average time complexity	$\boldsymbol{\Theta(n\ log\ n)}$
Best-case time complexity	$\boldsymbol{\Omega(n\ log\ n)}$
Worst-case space complexity	$\boldsymbol{O(n)}$

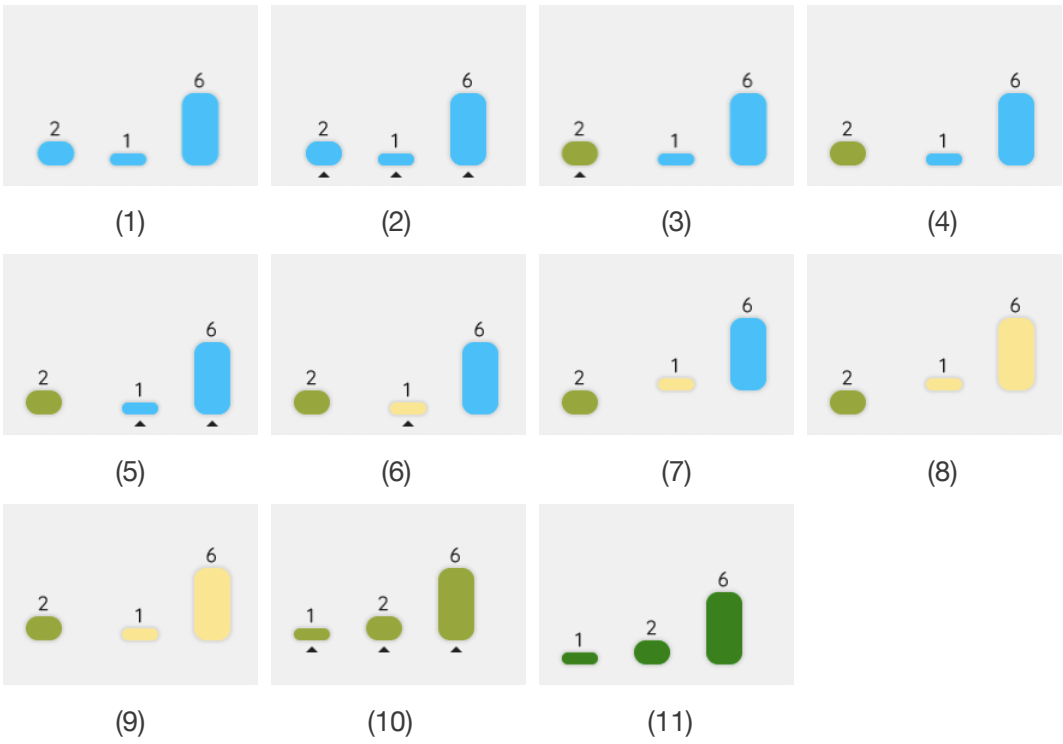
Pseudocode [3]

Algorithm: MergeSort(Arr)
Input: an array of integers Arr, the starting point of the left part **leftIndex**, the starting point of the right part **rightIndex**.
Output: The result of sorting Arr

```
if leftIndex > rightIndex then
  return Arr
else
  midIndex = (leftIndex + rightIndex) / 2
  mergeSort(array, leftIndex, midIndex)
  mergeSort(array, midIndex+1, rightIndex)
  merge(array, leftIndex, midIndex, rightIndex)
  return Arr
end if
```

Example

Sorting the list: 2,1,6



Implement in programming language

Java [1]

```java
public void mergeSort(int[] array){
        if(array == null) return;
        if(array.length > 1){
            int mid = array.length / 2;
            // Split left part
            int[] left = new int[mid];
            for(int i = 0; i < mid; i++){
                left[i] = array[i];
            }
            // Split right part
            int[] right = new int[array.length - mid];
            for(int i = mid; i < array.length; i++){
                right[i - mid] = array[i];
            }
            mergeSort(left);
            mergeSort(right);
            int i = j = k = 0;
            // Merge left and right arrays
            while(i < left.length && j < right.length){
                if(left[i] < right[j]){
                    array[k] = left[i];
                    i++;
                }else{
                    array[k] = right[j];
                    j++;
                }
                k++;
            }
            // Collect remaining elements
            while(i < left.length){
                array[k] = left[i];
                i++;
                k++;
            }
            while(j < right.length){
                array[k] = right[j];
                j++;
                k++;
            }
        }
    }
```

JavaScript [2]

```javascript
function mergeSort(arr) {
    var len = arr.length;
    if(len < 2) {
        return arr;
    }
    var middle = Math.floor(len / 2),
        left = arr.slice(0, middle),
        right = arr.slice(middle);
    return merge(mergeSort(left), mergeSort(right));
}

function merge(left, right)
{
    var result = [];

    while (left.length && right.length) {
        if (left[0] <= right[0]) {
            result.push(left.shift());
        } else {
            result.push(right.shift());
        }
    }

    while (left.length)
        result.push(left.shift());

    while (right.length)
        result.push(right.shift());

    return result;
}
```

C [2]

```c
void merge_sort_recursive(int arr[], int reg[], int start, int end) {
    if (start >= end)
        return;
    int len = end - start, mid = (len >> 1) + start;
    int start1 = start, end1 = mid;
    int start2 = mid + 1, end2 = end;
    merge_sort_recursive(arr, reg, start1, end1);
    merge_sort_recursive(arr, reg, start2, end2);
    int k = start;
    while (start1 <= end1 && start2 <= end2)
        reg[k++] = arr[start1] < arr[start2] ? arr[start1++] : arr[start2++];
    while (start1 <= end1)
        reg[k++] = arr[start1++];
    while (start2 <= end2)
        reg[k++] = arr[start2++];
    for (k = start; k <= end; k++)
        arr[k] = reg[k];
}

void mergeSort(int arr[], const int len) {
    int reg[len];
    merge_sort_recursive(arr, reg, 0, len - 1);
}
```

References

1. GeeksforGeeks Contributors (2018). *Java Program for Iterative MergeSort.* [Online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/java-program-for-iterative-merge-sort/ [Accessed 6 Mar. 2021].
2. Runoob Contributors (2018). *Merge sort*. [Online] Runoob. Available at: https://www.runoob.com/w3cnote/merge-sort.html [Accessed 6 Mar. 2021].
3. Visualgo Contributors (2014). [Software] Visualgo. Available at: https://visualgo.net/en/sorting [Accessed 6 Mar. 2021].
4. Wikipedia Contributors (2021). *Merge sort*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Merge_sort [Accessed 6 Mar. 2021].
5. Wikipedia Contributors (2021). *Space complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Space_complexity [Accessed 20 Mar. 2021].
6. Wikipedia Contributors (2021). *Time complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Time_complexity [Accessed 20 Mar. 2021].